Problema 3

Implementação

qr_GSP.m

O objetivo do pivoteamento parcial, assim como quando usado na fatoração LU, é melhorar a estabilidade numérica. Portanto, como implementada abaixo, essa função é uma versão melhorada de `qr_GSM`.

```
% Entradas:
%   A - matriz (m x n)
% Saídas:
%   Q = matriz (m x n) ortogonal
%   R = matriz (n x n) triangular superior
function [Q, R, P] = qr_GSP(A)
  [m, n] = size(A);
  Q = zeros(m, n);
  R = zeros(n, n);
  P = eye(n); % Matriz de permutação

  for j = 1:n
    % Encontra o índice da coluna com a maior norma
    [~, indice_max] = max(vecnorm(A(:, j:n)));
    indice_max = indice_max + j - 1;

    % Trocar colunas de A
    if indice_max ~= j
      A(:, [j, indice_max]) = A(:, [indice_max, j]);
      P(:, [j, indice_max]) = P(:, [indice_max, j]);
    end

    v = A(:,j); % j-ésima coluna de A

    % Obtém, por Gram-Schmidt, v o j-ésimo vetor de uma base ortogonal
    for i = 1 : j-1
      R(i,j) = dot(Q(:,i), v); % Usa o vetor atualizado
      v = v - R(i,j) * Q(:,i);
    end

    R(j,j) = norm(v,2);
    Q(:,j) = v / R(j,j); % j-ésimo vetor de uma base ortonormal
  end

end
```

Testes

Como pode ser visto os resultados para todas as matrizes são bem diferentes das questões anteriores, o que significa que foram feitas trocas de colunas, o pivoteamento entrou em ação.

```
>> [QPa, RPa, Pa] = qr_GSP(A)
QPa =
     0.5547    0.74125   -0.09759
     0.5547   -0.22237   -0.58554
    0.27735     -0.593   -0.19518
     0.5547   -0.22237    0.78072

RPa =
    3.6056    2.2188    0.27735
         0    1.0377     1.3342
         0         0     1.4639
```

```
>> [QPb, RPb, Pb] = qr_GSP(B)
QPb =
    0.82295   -0.37754    -0.36086   -0.41646
    0.25717    0.54622      0.4307          0
    0.46291    0.57836    0.034922   -0.36441
    0.20574   -0.47393     0.82648   -0.83293

RPb =
    19.442    18.516     10.904     10.595
         0    5.9281    0.69081     2.6669
         0         0     15.831     15.831
         0         0          0    2.1e-15
```

```
>> [QPc, RPc, Pc] = qr_GSP(C)
QPc =
    0.89443   -0.44721   -0.24254    NaN
    0.44721    0.89443    0.97014    NaN

RPc =
    4.4721    3.1305    2.6833      1.3416
         0    1.7889    0.89443    0.44721
         0         0    2.2e-16    1.1e-16
         0         0          0          0
```

Note que, assim como no problema anterior, aqui C apresenta valores NaN. Por isso, deixamos de fazer, nesse exemplo, os testes para essa matriz.

Agora resta verificar a acurárcia da fatoração e a ortogonalidade das matrizes resultantes.

```
>> norm((QPa * RPa) * Pa - A)
   2.2e-16

>> norm((QPb * RPb) * Pb - B)
   8.9e-16
```

```
>> norm((QPa' * QPa) - eye(3))
    3.7e-16

>> norm((QPb' * QPb) - eye(4))
    0.9413
```

Por fim, repetirei o teste para matrizes má condicionas feito para o problema anterior. No teste em questão, vemos que que o desempenho de `qr_GSP` é inferior ao de `qr_GSM`. Pois a Q_P está mais longe de ser ortogonal do qeu Q_M.

```
>> [QM ~] = qr_GSM(magic(20));
>> norm(QM'*QM - eye(20))
    0.99972
>> [QP, ~, ~] = qr_GSP(magic(20));
norm(QP'*QP - eye(20))
ans =
    0.99993

>> [QM ~] = qr_GSM(magic(100));
>> norm(QM'*QM - eye(100))
    0.99995
>> [QP, ~, ~] = qr_GSP(magic(100));
>> norm(QP'*QP - eye(100))
    1
```